VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Lukáš Stankovič

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Stankovič**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Shopsys s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Janoušek**

Konzultant bakalářské práce: Ing. Libor Plucnar

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

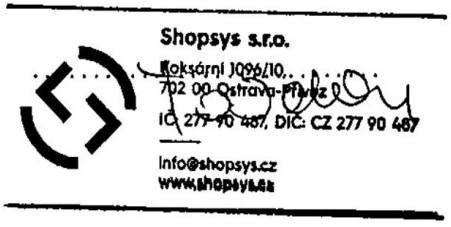
prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 23. dubna 2019



Rád bych poděkoval společnosti Shopsys za umožnění absolvování individuální odborné praxe. Také bych rád poděkoval veškerým kolegům z týmu, kterého jsem byl součástí.

Abstrakt

Tato bakalářská práce popisuje mé působení během odborné individuální praxe ve společnosti Shopsys s.r.o. Společnost se zabývá implementací velkých internetových obchodů na míru. Pracoval jsem na vývoji nových interních aplikací, které budou používány napříč všemi odděleními. Věnoval jsem se implementaci aplikace s interním názvem Všeaplikace sloužící jako jádro pro budoucí interní aplikace. Také jsem se věnoval aplikaci s názvem Agregátor. Veškeré tyto aplikace jsou naprogramovány v jazyku PHP. Měl jsem také na starost integrovat do projektu automatickou kontrolu standardů zdrojového kódu. Během praxe jsem uplatnil teoretické znalosti ze studia. Získané cenné dovednosti budu moci uplatnit například v budoucím podnikání, nebo v zaměstnání.

Klíčová slova: PHP, JavaScript, MySQL, e-shop, individuální odborná praxe, Shopsys, webové aplikace, databáze

Abstract

This bachelor's thesis deals with my individual professional practice in company Shopsys s.r.o. Company specializes on large-scaled customized e-commerce websites. I worked on developing new internal applications that will be used across all departments. I implemented an application internaly called Všeaplikace, which serves as an engine for future internal applications. I also developed an application internally called Agregátor. All these applications are programmed in PHP. I also integrated automatic standards check of the source code into the project. During my practice, I applied theoretical knowledge from my studies. I will be able to apply the acquired skills in future business or employment.

Key Words: PHP, JavaScript, MySQL, e-shop, Individual Professional Practice, Shopsys, Web applications, databases

Obsah

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface
B2B	–	Business-to-business
B2C	–	Business-to-consumer
CI	–	Continuous Integration
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
DTD	–	Document Type Definition
HTML	–	Hypertext Markup Language
HTTPS	–	Hypertext Transfer Protocol Secure
HTTP	–	Hypertext Transfer Protocol
JSON	–	JavaScript Object Notation
LESS	–	Leaner Style Sheets
PHP	–	Hypertext Preprocessor
PSR	–	PHP Standards Recommendations
REST	–	Representational State Transfer
SMS	–	Short Message Service
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
XML	–	Extensible Markup Language
XSD	–	XML Schema Definition

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Jelikož většina zaměstnavatelů v dnešní době uvítá kromě teoretických i praktické znalosti, rozhodl jsem se vypracovat bakalářskou práci jako individuální odbornou praxi. Díky této volbě jsem mohl propojit veškeré nabyté teoretické znalosti za celou dobu studia s praktickými zkušenostmi a seznámit se s agilním vývojem softwaru, či poprvé spolupracovat v týmu pěti lidí vedeným projektovým manažerem.

Tato práce popisuje mé působení v ostravské firmě Shopsys, zabývající se vývojem velkých profesionálních e-shopů. Za celé své působení jsem pracoval především v programovacím jazyku PHP a JavaScript a s databázemi v jazyku MySQL. Měl jsem možnost se podílet na vývoji nových interních aplikacích s interním názvem Všeaplikace a Agregátor. Tyto aplikace se budou používat napříč skoro všemi odvětvími ve společnosti Shopsys. Během vývoje interních aplikací jsem si vyzkoušel jednak analýzu problému, ale i kompletní implementaci a několikrát mít možnost udělat kolegovi revizi kódu neboli tzv. Code Review. Účastnil jsem se i školení ohledně různých nejnovějších technologií používaných v oblasti vývoje webových aplikací. Mezi ně například patří PHP framework Symfony, práce s NoSQL databází MongoDB, napojení aplikace na Elasticsearch pro přesnější vyhledávání a další.

2 O společnosti Shopsys

Společnost Shopsys s.r.o. vznikla v roce 2003 již při středoškolských studiích zakladatele Petra Svobody. V začátcích svého fungování se firma orientovala na malé a krabicové řešení e-shopů. V roce 2005 již společnost realizovala více než 100 e-shopů a o pět let později Shopsys již slavil 500 realizovaných e-shopů. Momentálně se společnost soustředí na realizaci velkých B2C a B2B e-shopů. Společnost se také věnuje například výkonnostnímu marketingu, technické analýze, testování, tvorbě designu nebo drátěného modelu webové stránky. Velké úsilí firma vkládá do open source projektu Shopsys Framework. Mezi známé realizované e-shopy patří například Okay, nadnárodní B2B e-shop s kancelářskými potřeby OfficeDepot nebo e-shop s chovatelskými potřebami Superzoo [1]. Veškeré e-shopy jsou postaveny na jazyku PHP.

Hlavní pobočka firmy se nachází v Ostravě. Další menší pobočky se nacházejí v Pardubicích, Praze a na Slovensku v Žilině. Společnost čítá celkově zhruba 80 zaměstnanců.

2.1 Shopsys Framework

V posledních dvou letech se firma soustřeďuje na vydávání volně dostupného frameworku Shopsys Framework, který má za cíl ulehčit práci ostatním, nejen českým agenturám a programátorům ve vývoji velkých e-shopů s mnoha individuálními úpravami. Shopsys Framework je postaven především na programovacím jazyku PHP a jeho frameworku Symfony. Je možné jej nainstalovat na operační systémy Linux, macOS, či Windows. Shopsys Framework je také připraven na technologii pro virtualizaci Docker. Díky své architektuře je lehce aktualizovatelný [1].

2.2 Pracovní zařazení

Před nástupem na individuální odbornou praxi jsem nejprve musel projít přijímacím řízením, které se skládalo z programovacího testu v jazyku PHP a osobního pohovoru. Pár dní po absolvování programovacího testu mě kontaktovala HR manažerka, která mě pozvala na již zmíněný osobní pohovor. Součástí pohovoru byly otázky týkající se znalostí v oboru a mě samotného. Ihned poté jsem se potkal s projektovým manažerem týmu, do kterého jsem měl být zařazen a mohl s ním pohovořit o fungování týmu, používaných procesech a nástrojích. V rámci odborné praxe jsem byl zařazen do týmu, který vyvíjí veškeré interní aplikace a nastoupil na pozici junior programátor.

První den nástupu jsem byl proveden po celé firmě. Byli mi také představeni jednotliví spolupracovníci z různých týmů. Následně mi zkušený senior programátor pan Plucnar, který mě má jako odborný konzultant po celé mé působení na starost, ukázal veškeré nástroje používané v týmu, způsob, jak se přihlásit do docházky, jakým způsobem lze komunikovat s ostatními spolupracovníky a další základní věci. Současně jsem měl vyhrazený čas na prozkoumání a pročtení například programátorských standardů, jakým způsobem pracovat s Gitem nebo například jakým procesem je zapracována nová úprava.

Další dny byly ve znamení školení na veškeré procesy, které ve firmě existují. Absolvoval jsem školení na různé stávající interní aplikace, které se používají například na měření času stráveném na daném úkolu či aplikaci pro možnou budoucí komunikaci s klienty. Dále jsem prošel tréninkovým programem, kde byly vyzkoušeny testovací úpravy na zkušební interní aplikaci. Také jsem si nanečisto díky testovacím implementacím vyzkoušel veškeré pracovní postupy včetně testování a validace úpravy. Během testovacích úprav bylo nachystáno několik problémů, se kterými jsem si musel poradit. Mezi ty patřily například vyřešení konfliktů při slučování úprav či vymyšlení řešení daného algoritmu. Během své odborné praxe jsem se podílel jen na vývoji nových interních aplikací.

2.3 Procesy

Vzhledem k tomu, že jsem nikdy před působením na praxi nepracoval v týmu vícero lidí, byly pro mě určité procesy používané pro vývoj softwaru nové. Zde jsem například uplatnil teorii z předmětu *Úvod do softwarového inženýrství* ze třetího semestru a *Vývoj informačních systémů* z pátého semestru. Ve společnosti se pro vývoj používá agilních metodik. Znamenalo to například, že jsme pravidelně měli takzvaný *standup*, při kterém jsme každý z týmu řekli, co budeme daný den dělat a jaké problémy budeme řešit. Dále jsme v pravidelných intervalech měli delší poradu nazývanou *retrospektiva*. Při této poradě jsme se zamýšleli nad uplynulým obdobím. Konkrétně nad čtyřmi kategoriemi – co se nám jako týmu povedlo a nepovedlo, co nám může v budoucím období pomoct a popřípadě ohrozit.

Dalším typem porady byla předimplementační schůzka. Na tomto typu porady se řešila správná implementace či řešení určitého problému. Ve většině případů probíhala s mým odborným konzultantem panem Plucnarem. Během této porady byl vymyšlen následný postup a teoretické řešení dané úpravy. Tento proces byl velmi užitečný, jelikož jsem se díky němu naučil mnoho nových možností, jak k problému přistupovat.

Byl jsem také seznámen s procesem používání verzovacího systému Git. Celou dobu jsem na interních aplikacích pracoval tak, aby při nové úpravě byla vytvořena nová větev z hlavní větve jménem *master*. Nová větev vždy začínala dle mých iniciál. Poté následoval název úpravy, případně číslo chyby, kterou jsem opravoval. Jakmile byla úprava implementována, vytvořil jsem ve webovém prostředí Gitlab Merge Request. Zde mi udělal můj odborný konzultant revizi kódu neboli Code Review a vysvětlil mi, co jsem udělal špatně, či případně co je vyřešeno správně. Revizí kódu se jednak předejde syntaktickým chybám, ale také se daný programátor, který revizi dělá, může naučit novým způsobům, jak daný algoritmus či problém řešit. Následně byla úprava předána na oddělení testerů, kde je manuálně testery otestována. Jakmile byla úprava úspěšně otestována, mohla být začleněná pomocí tzv. *merge* do hlavní *master* větve.

3 Seznam zadaných úkolů

Během mé odborné praxe jsem měl za úkol pracovat na nové interní aplikaci s interním názvem *Všeaplikace*, která funguje jako hlavní jádro pro ostatní menší interní aplikace. Dalším úkolem bylo pracovat na interní aplikaci *Agregáror*.

3.1 Všeaplikace

Kód a technologie všech interních aplikací byly natolik rozdílné, že je bylo těžké udržovat. Některé interní aplikace již byly také zastaralé a nevyhovovaly potřebám. Díky tomu vzplanul plán na sjednocení těchto aplikací do jedné velké interní aplikace. Ze začátku bylo za úkol pracovat na této kompletně nové interní aplikaci, která má sloužit jako hlavní jádro pro další menší interní aplikace. Mezi takové aplikace patří Agregátor, nová docházka zaměstnanců, počítadlo nákupů ve firemním bufetu a další. Z důvodu, že aplikace bude sloužit jako jádro pro ostatní interní aplikace, bylo velmi důležité, jakou bude mít *Všeaplikace* architekturu a jak bude koncipována. Aplikace v rámci menších aplikací bude obsahovat citlivá data. Tím pádem musí být jádro maximálně zabezpečené. Časový předpoklad čisté mé programátorské práce byl odhadnut na 200 hodin.

3.2 Agregátor

Jakmile byla vytvořena *Všeaplikce*, bylo třeba pracovat na nové interní aplikaci *Agregátor*, která má za úkol agregovat veškerá důležitá data pro chod a správu e-shopu. Agregátor již běží na jádru předchozí *Všeaplikace*. Měl jsem za úkol například vytvořit aplikaci Diff Tool, která má na starost kontrolovat prvky na definovaných stránkách. V případě změny na dané stránce zaznamená jaká změna nastala a tuto změnu odešle administrátorovi na e-mail. V rámci další zadané aplikace Agregátoru bylo vytvořit napojení e-shopů na Agregátor. V rámci tohoto napojení se budou přeposílat informace o běhu e-shopu a z nich budou v budoucnu generovány různé statistiky. Dále se budou přeposílat důležitá data týkající se nastavení e-shopu. Náročnost všech mých programátorských prací na aplikaci Agregátor byla odhadnuta na 100 hodin.

4 Používané technologie

Díky bakalářské praxi jsem se mohl naučit či se zdokonalit ve velkém množství různých technologiích používaných v rámci vývoje internetových aplikací. Jedná se zejména o níže zmíněné technologie.

Nejpoužívanější jazyk ve firmě je PHP. Je to skriptovací programovací jazyk určený pro vytváření dynamických webových aplikací. Veškeré skripty jsou vykonávány na serveru a uživateli jsou přenášeny pouze výsledky těchto skriptů. Syntaxe jazyka vychází ze známých jazyků jako například C, Java, nebo Pascal. Jazyk PHP podporuje jednak procedurální přísup, ale i objektově orientovaný přístup [2]. V tomto jazyce jsou napsány veškeré hlavní funkčnosti interních aplikací, na kterých jsem během své odborné praxe pracoval a v tomto jazyce se setkal pouze s objektově orientovaným přístupem.

Pro databáze je v interních aplikacích nejčastěji používáno MySQL. Je to systém řízení báze dat pracující na relačním databázovém modelu. Komunikace s databází probíhá pomocí jazyka SQL. MySQL je často používáno ve webových aplikacích. Používá několik formátů pro ukládání dat, nejznámější z nich jsou InnoDB a MyISAM. MyISAM oproti InnoDB například uzamyká celé tabulky, a ne pouze řádky, nepodporuje transakce a cizí klíče [3]. Během působení na praxi byla veškerá data ukládána právě do tohoto systému s formátem InnoDB.

Pro tvorbu struktury webové stránky je nejčastěji používán značkovací jazyk HTML [4]. Pro vývoj interních aplikací ve firmě je HTML používán v kombinaci se šablonovacím systémem Smarty. Tento šablonovací systém umožní vypisovat hodnoty proměnných přiřazených pomocí PHP. Pro samotný vzhled webové stránky jsou používány kaskádové styly CSS. Ve společnosti je používán preprocesor LESS, který doplní CSS o řadu užitečných funkcí, jako jsou proměnné, funkce či zanořování jednotlivých deklarací [5].

Další používanou technologií je JavaScript. Je to multiplatformní skriptovací jazyk, jehož syntaxe patří taktéž do rodiny jazyků jako C či Java. Oproti jazyku PHP se JavaScript obvykle spouští po stažení webové stránky v internetovém prohlížeči uživatele [6]. Pro JavaScript existuje nespočetné množství různých knihoven a frameworků, které usnadňují práci programátora. V rámci praxe byl JavaScript používán s knihovnou jQuery [7]. S pomocí té lze jednoduše realizovat například AJAX či manipulovat s obsahem webové stránky. Během praxe bylo v tomto jazyce řešeno například skrývání či zobrazování určité části stránky. Pomocí jQuery jsou realizovány animace nebo validace formulářů na straně uživatele.

Veškeré verzování zdrojového kódu je spravováno pomocí systému Git. Systém používá pro verzování adresářové stromy se soubory [8]. Z důvodu lepší správy jsou repozitáře spravovány ve webovém prostředí Gitlab a zde také probíhala například kontrola kódu tzv. Code Review. Při vývoji byl nejčastěji Git používán v rámci vývojového prostředí PhpStorm nebo pomocí příkazů v příkazovém řádku. Pro každou úpravu, která byla realizována byla vytvořena větev, která vycházela z hlavní nazývající se *master*. Jakmile úprava prošla Code Review a manuálním testováním, byl proveden tzv. *merge* do hlavní větve *master*.

5 Všeaplikace

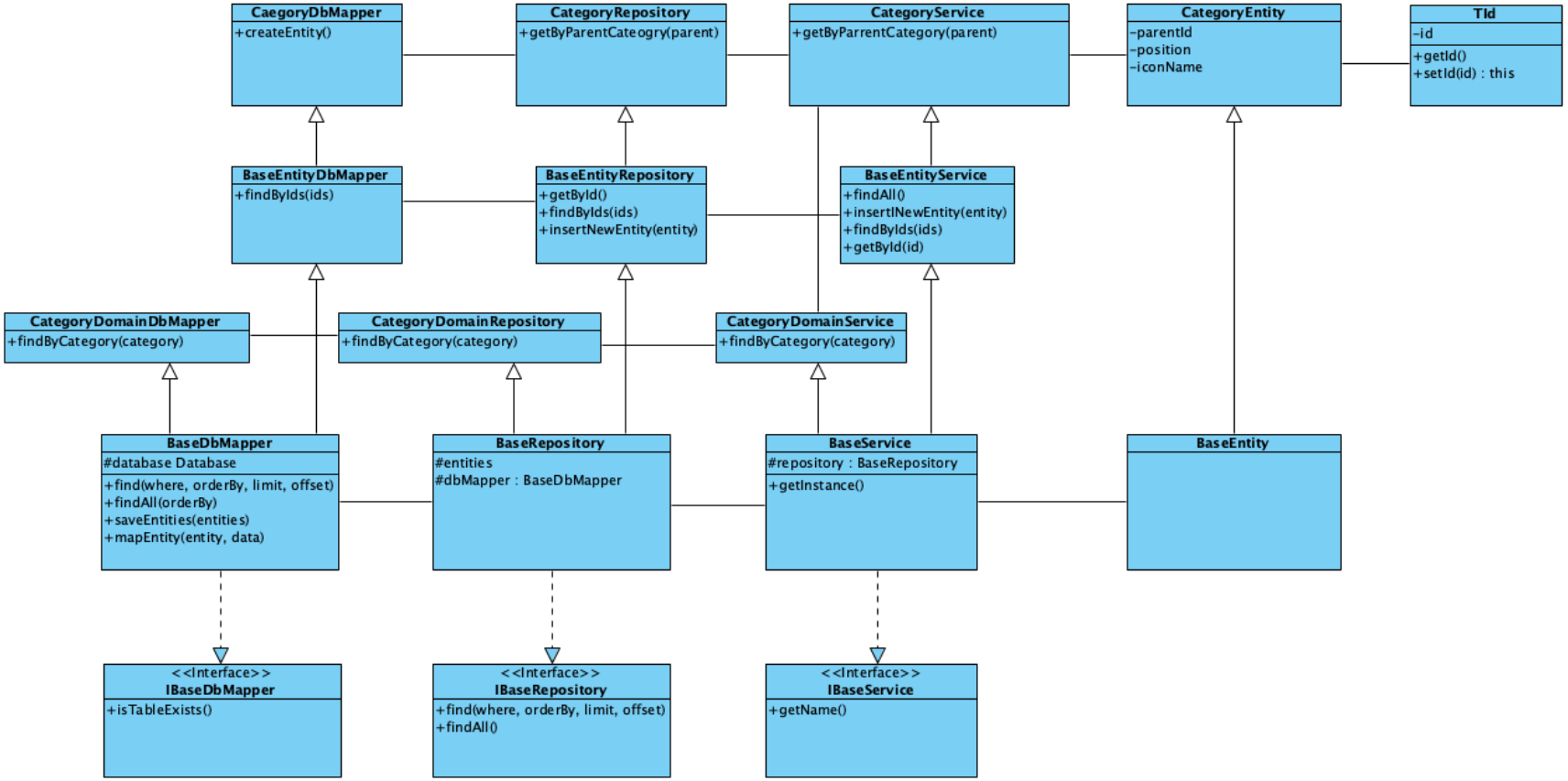
Během absolvování své odborné praxe na interních aplikaci jsem spolupracoval s druhým studentem VŠB. Celou dobu na nás dohlížel hlavní architekt pro interní aplikace a na mě samotného dohlížel odborný konzultant práce Libor Plucnar. Tato aplikace slouží jako základ pro ostatní menší interní aplikace.

5.1 Jádro aplikace

Jelikož je nutné, aby byla aplikace lehce rozšířitelná, bylo třeba vymyslet architekturu celého jádra, na kterém v budoucnu budou veškeré další menší interní aplikace postaveny. Základ celého jádra je postaven na řešení, které se ve firmě používá pro stavbu e-shopů. Tento základ byl v prvních fázích vývoje upraven tak, aby byl vyhovující pro interní aplikaci. Nejprve byly odstraněny námi nepotřebné komponenty, jako byl například košík či produkty. Jádro aplikace bylo následně upraveno tak, aby bylo kompatibilní s novějším PHP 7.

Celý systém je postaven na vícevrstvé architektuře, kdy má každá tabulka v databázi vlastní entitu a je k ní přistupováno pomocí vlastní servisní vrstvy, která volá repozitář s databázovým mapperem. Servisní vrstva je jediná, která může být programátorem volána. Ostatní vrstvy jsou volány pouze ze sousedních vrstev. Druhá vrstva je repozitář, který zajišťuje mezipaměť pro načtené entity. Pokud daná hledaná entita není načtená v seznamu entit, je zavolán databázový mapper, který repozitáři vrátí danou entitu či seznam entit. Jedná se o návrhový vzor *Identity Map*. Tento vzor byl vybrán z důvodu ušetření dotazů na databázi, které jsou zároveň nejpomalejší článek při vykreslení stránky.

Základním kamenem architektury jsou rozhraní, kterými jsou definovány základní veřejné metody. Z těchto rozhraní vycházejí abstraktní třídy s předponou *Base*. Tyto třídy obsahují definice základních metod pro manipulaci s databází a s jednotlivými objekty. Dále z těchto tříd dědí další abstraktní třídy s předponou *BaseEntity*, anebo již konkrétní třídy. Pomocí tříd dědicích přímo *Base* tříd je manipulováno s databázovými objekty neobsahující sloupec *id*. Nejde tedy poté o entity, ale jen o určité propojovací objekty. Pro entity byly vytvořeny již zmíněné abstraktní třídy s předponou *BaseEntity*. Obsahují veškerou funkčnost jako *Base* třídy, avšak přidávají k nim metody potřebné pro práci s entitami. Obsahují navíc například metody nutné pro mapování entit a mezipaměť pro načtené entity. Z těchto tříd dědí již skutečné třídy s konkrétními entitami. Samotná entitní třída obsahuje pouze stejné atributy jako v databázi a její *gettery* a *settery*. Jednotlivé databázové sloupce jsou definovány ve stylu *snake_case*. Atributy v entitních třídách jsou definovány ve stylu *camelCase*. Samotné mapování probíhá v databázovém mapperu automatickým přepisem mezi těmito dvěma styly zápisu. Celá architektura společně s částí tříd kategorií je vizualizovaná na obrázku 1.



Obrázek 1: Základní architektura

V minulosti se stávalo, že každý programátor používal vlastní rozdílné označení pro atributy třídy stejného významu napříč celou aplikací. Bylo tedy nutné v co největší míře sjednotit názvy atributů a metod všech entit. Pro tyto účely byly zvoleny *traity* a ty se na rozdíl od rozhraní například liší tím, že obsahují navíc kompletní definice metod. Traity jsou vytvořeny na často používané atributy jako například *id*, *name* a další. Pro každý takový atribut je zároveň definován *getter* a *setter*. Každá třída může zároveň používat vícero traitů. Výhoda jejich použití je také v tom, že třída neobsahuje zbytečné řádky navíc, ale na první pohled je viditelné, že třída má určitý atribut, u kterého je jisté, že používá stejné metody jako v jiné části aplikace. Veškeré *traity* jsou uloženy v jednom adresáři. Definici *traitu* pro atribut *id* je možné vidět ve výpisu zdrojového kódu 1.

Pro veškeré *settery* jsou napříč aplikací využívány takzvané *fluent settery*. Ty se vyznačují tím, že ihned po nastavení hodnoty atributu vrací celý objekt a to má výhodu, že je možné volat několik těchto metod za sebou jako jeden příkaz. Nastavování hodnot daného objektu tedy probíhá řetězením těchto *setterů*. Fluent setter je možné vidět ve výpisu 1 u metody *setId*.

Jazyk PHP podporuje klíčová slova pro datové typy omezeně. Datový typ není například definován u atributů třídy. Z toho důvodu jsou napsány pro veškeré atributy a metody tříd anotace, které jsou užitečné nejen pro programátora, aby na první pohled viděl datový typ proměnné, ale i pro vývojové prostředí, které díky anotacím dokáže lépe napovídat při psaní zdrojového kódu. Anotace jsou zapisovány jako víceřádkový komentář na základě dokumentace *PHPDoc*. Ačkoli je na aplikaci používáno PHP 7 nabízející datové typy v deklaracích metod, nejsou používány z důvodu zpětné kompatibility se starším PHP 5.6. Od verze PHP 7 je také možné využívat návratové hodnoty metod a ty nejsou v interní aplikaci z důvodu zpětné kompatibility používány. Zpětná kompatibilita je důležitá z toho důvodu, aby bylo možné veškeré existující interní aplikace spustit na jednom serveru.

```
trait TId {

   /**
    * @var int
    */
   private $id = 0;

   /**
    * @return int
    */
   public function getId() {
      return $this->id;
   }

   /**
```

```
     * @param int $id
     * @return $this
     */
    public function setId($id) {
        $this->id = (int)$id;
        return $this;
    }

}
```

Výpis 1: Trait pro atribut id

Aplikace je kompletně objektově orientovaná. Celé jádro aplikace je napsáno v jazyku PHP. Dále je zde použitý jazyk JavaScript, pomocí kterého je řešen například AJAX či animace. Veškeré styly jsou napsány v jazyce CSS s preprocesorem LESS, struktura stránek pak pomocí značkovacího jazyka HTML a šablonovacího systému Smarty.

5.2 Realizace subdomén

Aplikace je stavěná jako základ pro ostatní menší interní aplikace. Tato funkčnost je realizována pomocí subdomén, které jsou uskutečněny na základě složek s předponou *domain_*. Veškeré základní soubory jsou v adresáři *domain_common*. Tyto soubory lze přetížit tím, že je vytvořen soubor v dané subdoméně ve stejné adresářové struktuře. Programátor pro ostatní interní aplikace tedy nemusí veškerou základní funkčnost programovat znova. Pro Agregátor, na kterém jsem pracoval jako na druhém projektu, existuje například složka *domain_aggregator* obsahující pouze soubory důležité pro samotnou funkčnost agregátoru.

5.3 Oddělení

Mým úkolem bylo navrhnout a realizovat funkčnost oddělení jednotlivých stránek, kdy je jedna stránka zobrazena jako jedno oddělení. Pod odděleními je možné si představit kategorie pro jednotlivé stránky. Jelikož byl požadavek mít veškerá oddělení přístupná na více subdoménách a v různých jazycích, jsou v databázi uloženy jako tři propojené tabulky – hlavní tabulka se základními informacemi o odděleních, tabulka s překlady a tabulka se zařazením do domén. Každá z nich je realizována dle architektury jádra vlastními modely, které mají vlastní entitu, repozitář, servisní vrstvu a databázový mapper.

5.3.1 Export a import oddělení

Aby byla zajištěna stejná oddělení a stránky pro všechny kdo na projektu pracují, byl realizován export a import těchto oddělení. Požadavek byl také na efektivní zobrazení exportovaných kategorií v *Merge Requestu* v prostředí Gitlab a v co největší míře zabránění *merge konfliktům*,

kdy najednou vícero lidí přidá v administraci různá oddělení. Řešení těchto požadavků spočívá v tom, že je každé jednotlivé oddělení exportováno do samostatného souboru, kdy se v názvu souboru zároveň nachází ID daného oddělení. Dalším procesem, kterým se předejde konfliktům je, že oddělení je vždy programátorem vytvořeno pouze na ostrém serveru a je z ostrého serveru exportováno a následně naimportováno na lokální prostředí programátora, kde exportovaná oddělení *commitne*. Po tomto procesu jsou oddělení jak na ostré verzi, tak v repozitáři verzovacího systému *Git*. Při nahrávání úpravy na ostrý server jsou veškerá stará oddělení smazána a přepsána novými ze souborů.

Nejdříve ale bylo nutné vybrat jakým způsobem a v jakém formátu budou oddělení exportována. Bylo prozkoumáno několik různých formátů, mezi které patří například XML, JSON, CSV, Yaml a další. Dále do užšího výběru byly zvoleny formáty XML a JSON a bylo zjištěno, že výhoda JSONu oproti XML je například v rychlém zpracování souboru při malém počtu dat. Další výhoda JSONu je v jednodušším zpracovávání pomocí jedné metody například v jazyce JavaScript či PHP. Naopak výhoda XML spočívá v lepší čitelnosti a také v lepším zobrazení změny v *commitu*, kdy je při revizi kódu jednodušší poznat, která část souboru byla změněna. Soubor je lépe programátory zapisovatelný. Nakonec bylo dohodnuto na použití XML právě kvůli čitelnosti.

Jakmile byl vybrán formát souboru pro export, bylo nutné se rozhodnout, zda a jak se bude XML soubor validovat. Zde se nabízela jednak varianta XML Schema Definition (XSD), nebo Document Type Definition (DTD), anebo použití validace na straně aplikace. Výhoda XSD oproti DTD je, že podporuje datové typy a jmenné prostory. Ačkoli XSD používá XML syntaxi je složité tento jazyk pochopit. Z důvodu složitosti a časové náročnosti bylo nakonec rozhodnuto XML soubor nevalidovat v rámci existujícího schémata, ale na úrovni mapování do entity. Tento krok byl také zvolen z toho důvodu, že XML není importováno ze žádné třetí služby, ale generováno automaticky v rámci aplikace. Nemělo by tedy k chybám struktury docházet. Pokud by bylo XML nevalidní, tak by nedošlo ke správnému namapování oddělení na databázovou entitu. Špatně namapovaná entita by se vůbec do databáze nevložila a nezpůsobila by žádné následné problémy.

Pro generování XML je možné použít vícero způsobů. Jedna z nejsnazších metod je pomocí Smarty, kde si programátor vytvoří XML šablonu s proměnnými. Ty jsou do šablony doplněny při generování. Další snadný způsob je s použitím třídy SimpleXML. Tento způsob například nabízí přístup v PHP k prvkům XML pomocí řetězení prvků. Nakonec bylo použité nejkomplexnější řešení pomocí PHP třídy *DOMDocument*. Tato třída je realizována přímo na standardu W3C DOM API. Obojí řešení například nabízí efektivní přidání XML tagu do již existujícího XML souboru, objektový přístup nebo účinné hledání prvku pomocí XPath. Při generování jsou vytvářeny XML prvky pomocí vzájemně propojených objektů. Samotný export oddělení neprobíhá v administraci, ale v instalátoru, který je zároveň automaticky spouštěn při nasazování úpravy na ostrý server. Soubor s jedním exportovaným oddělením je možné vidět na výpisu 2.

Pro načítání při importu je pro vyhledávání samotných prvků s odděleními využíván jazyk

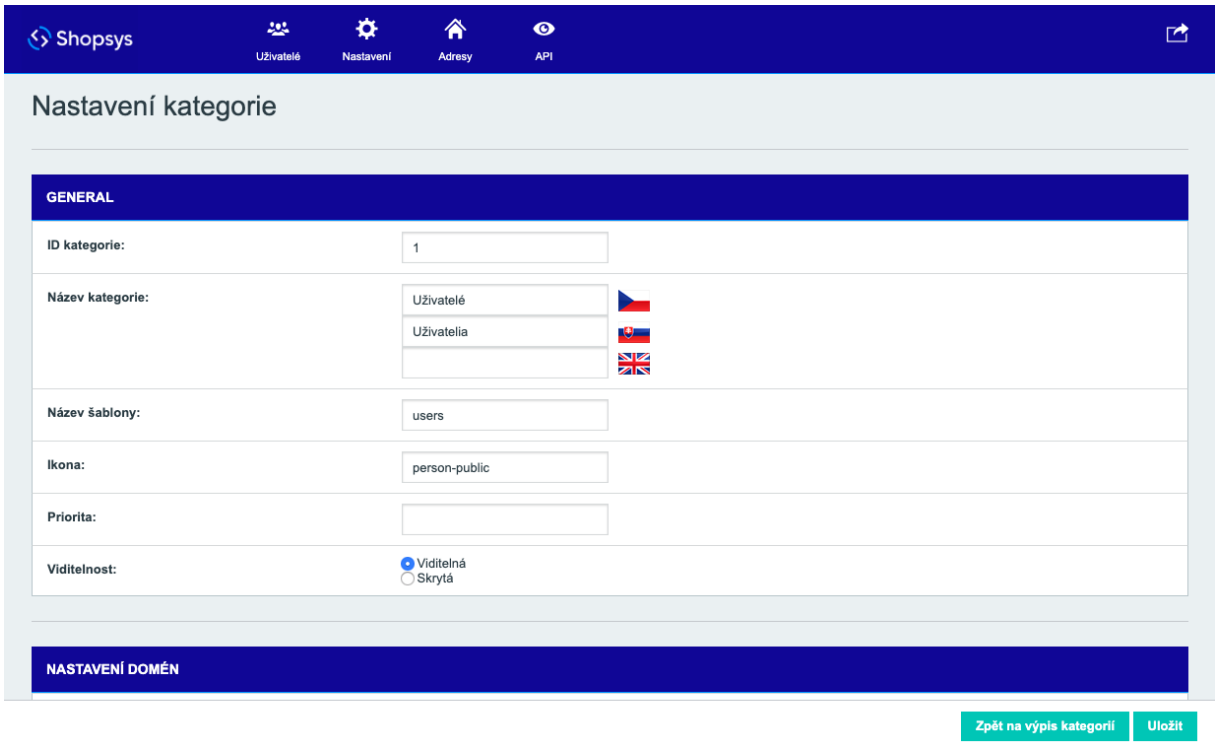
XPath. Je to jazyk, pomocí kterého je možné načítat prvky XML dokumentu [9]. Prvek v XML dokumentu je možné nalézt například dle cesty uzlů, která je oddělená pomocí lomítek. Nalezené kategorie jsou poté mapovány na pole entit, které jsou v dávkách uloženy do databáze. Pomocí dávek je do jednoho dotazu zakomponováno maximálně 250 entit. Omezení je nastaveno z důvodu omezení délky jednoho dotazu. Tím, že jsou kategorie uloženy pomocí dávek, zvýší se rychlost importu.

```
<?xml version="1.0" encoding="UTF-8"?>
<category>
    <id>1</id>
    <parent_id>0</parent_id>
    <level>0</level>
    <is_hidden_category>0</is_hidden_category>
    <priority>0</priority>
    <icon_name>person-public</icon_name>
    <template_name>users</template_name>
    <translations>
        <translate>
            <language_id>1</language_id>
            <name>Užívatelé</name>
        </translate>
        <translate>
            <language_id>2</language_id>
            <name>Užívatelia</name>
        </translate>
    </translations>
    <domains>
        <domain>
            <id>1</id>
        </domain>
        <domain>
            <id>2</id>
        </domain>
    </domains>
</category>
```

Výpis 2: Exportované oddělení

5.3.2 Nastavení odddělení

Pro správu oddělení byla vytvořena stránka v administraci. Na této stránce mohou být oddělení uživatelem s danými přístupovými právy přidávána, upravována či smazána. Administrátorem mohou být u oddělení nastaveny různé vlastnosti a mezi ty například patří vícejazyčné názvy oddělení, priorita, viditelnost, název šablony, se kterou se má oddělení propojit nebo název SVG ikony, která je vykreslena v hlavní navigaci. Stránku s nastavením oddělení je možné vidět na obrázku 2.



Obrázek 2: Administrace - nastavení oddělení

Spolu s vytvořením administrace bylo nutné vymyslet způsob automatického zobrazování překladových formulářových prvků. Jelikož jsou veškeré formulářové prvky generovány automaticky pomocí šablonovacího systému, bylo třeba dodělat doplněk do Smarty. Doplněk zajistí, aby bylo pro vícejazyčný formulářový textový prvek vygenerováno tolik polí, kolik existuje nastavených jazyků. Pro vypsání těchto polí existují dva možné způsoby. První způsob spočívá v zobrazení vícejazyčných formulářových prvků jako pole. Prvky mají stejný název. Tvar názvu prvků je následující – *category_name[X]*, kde *X* je ID jazyka. Druhý způsob spočívá v zobrazení vícejazyčných textových prvků jako samostatných a oddělených formulářových prvků. Každý prvek pro každý jazyk má jiný název ve tvaru *category_name_____X*, kde *X* je ID jazyka. Výhoda prvního řešení je v tom, že je tento způsob v HTML podporován ve výchozím stavu. Nevýhoda je, že s tímto řešením nefunguje validace formulářových prvků pomocí speciálního doplňku SmartForm na straně serveru pomocí PHP. Z toho důvodu byla nakonec zvolena druhá možnost

s tím, že po odeslání formuláře a následné validaci se formulářové prvky přeformátují na pole prvků se stejným názvem. Po odeslání formuláře jsou textové formulářové prvky zpracovány stejně jako v první možnosti.

Aby bylo do budoucna ostatním vývojářům ušetřeno co nejvíce času při programování formulářů, byla vytvořena metoda, pomocí které je celý formulář namapován na předem definované entity. Veškeré entity jsou hromadně vloženy do databáze. Stejný princip je využit i při zobrazování již vyplněného formuláře, kdy je třída, na kterou má být formulář namapován, předem programátorem nadefinována. Jedinou podmínkou pro správnou funkčnost je, aby formulářové prvky měly stejné názvy jako atributy v dané třídě. Výhoda tohoto řešení je také v tom, že jsou veškeré formuláře řešeny jednotně.

5.3.3 Technická reprezentace oddělení

Oddělení jsou uložena v relační databázi jako řádky obsahující odkaz na nadřazené oddělení. Bylo nutné vymyslet nejlepší způsob vytváření stromu používaného pro vykreslení těchto jednotlivých oddělení.

Nabízelo se zde využití rekurze, kdy by potomci jednotlivých oddělení byli pomocí rekurze ukládáni do atributů daných entit. Další způsob byl oddělení reprezentovat jako vícerozměrné pole, kde by klíč u potomků bylo jejich nadřazené oddělení. Bylo zjištěno, že využití rekurze je nejjednodušší implementovatelný způsob zobrazení stromu, avšak velká nevýhoda je pomalé načítání dat. Další způsob spočívá v tom, že strom nebude nikdy načítán jako celek, ale budou načítány pouze jeho části. Výhodou řešení je, že je vždy načítán pouze lineární seznam pro jedno nadřazené oddělení. Další výhoda spočívá v tom, že jsou různé části načítány, pokud je to potřeba. Z důvodu optimalizace rychlosti a lepšímu přístupu k datům jsou překlady načítány vždy pro jednu danou doménu a daný jazyk. Výjimkou je pouze administrace, kde je třeba vidět veškeré překlady daného oddělení.

Oddělení jsou tedy nakonec reprezentována jako jednorozměrné pole entit. Aby bylo při načítání zabráněno složitému dotazu na databázi a tím pádem následnému vytváření dočasných tabulek, byla zvolena cesta dvou elementárních databázových dotazů. Výsledky těchto dotazů jsou následně zpracovány až v PHP. Načítání oddělení z databáze funguje tak, že nejprve jsou do pole načtena pouze jednotlivá ID oddělení, která jsou na dané doméně povolena. Na základě tohoto pole jsou do dalšího jednorozměrného pole načítány jednotlivé entity s veškerými atributy. Obsah tohoto pole je následně vykreslován do stránky.

5.3.4 Vykreslování oddělení

S odděleními je úzce spojené vykreslování navigace a stránek. Cílem bylo, aby pokud někdo přidá nové oddělení, byla vytvořena navigace navázána na definované šablony. Jelikož oddělením lze nastavit různou úroveň vykreslení, byla nejprve vytvořena hlavní vrchní navigace. V té jsou zobrazeny pouze oddělení s nulovou úrovní. Pokud existují oddělení podřízené pro dané hlavní

oddělení, jsou zobrazeny v boční levé navigaci při vykreslení stránky. Pokud pro hlavní oddělení neexistují žádná pododdělení, boční navigace není vůbec vykreslena. Logika vykreslení oddělení je řešena v jazyce PHP a samotné vykreslování je řešeno na úrovni šablony.

Z důvodu optimalizace jsou další úrovně načítány pouze při kliknutí na ikonu pro zobrazení u nadřazeného oddělení. Oddělení jsou načítána pomocí technologie AJAX. Pokud tedy uživatel klikne na tlačítko rozevření pododdělení, je odeslán HTTP požadavek, který v databázi nalezne odpovídající oddělení pro dané nadřazené oddělení. Díky technologii AJAX se nemusí pro zobrazení navigace daného pododdělení načíst celá HTML stránka znovu. Načtená oddělení jsou vložena do obsahu stránky, aby se již nemusela na jedné stránce načítat vícekrát. Pokud jsou tedy načtená pododdělení uživatelem skryta či znovu pomocí kliknutí na ikonu pro rozevření odkryta, není odeslán již žádný HTTP požadavek. Oddělení jsou pouze z obsahu stránky pomocí JavaScriptu odkryta, nebo skryta. Tímto způsobem je zobrazování navigace mnohonásobně urychleno, jelikož nejvíce času pro zobrazení navigace zabere namapování dat do jednotlivých entit. Zároveň je také ulehčeno databázovému serveru. Tento návrhový vzor se nazývá *Lazy Loading*. Účelem tohoto vzoru je, aby požadována data byla načítána pouze, pokud je to potřeba.

Další hlavní požadavek byl na čisté URL stránek. Čistá URL je taková, že neobsahuje žádné informace o souboru či další citlivé údaje. Opakem čistých URL je dynamická URL. Příklad takové URL článku blogu s ID 3 a názvem Lorem Ips je například *www.example.com/blog.php?id=3*, naopak příklad čisté URL je *www.example.com/blog/lorem-ips*. Pro tyto účely byla vytvořena databázová tabulka *url*, která uchovává samotnou URL stránky, název šablony, kterou má stránka vykreslit a ID oddělení, na které má být stránka odkázána. Dále byla vytvořena speciální doplňková metoda *link_category* do šablonovacího systému, která po zadání ID oddělení vypíše čistou URL pro zadané oddělení.

5.4 Kontrola zdrojového kódu

Jelikož při vývoji nové velké interní aplikace sloužící pro různá oddělení ve společnosti narůstaly problémy s dodržováním standardů psaní zdrojového kódu, bylo nutné proces kontroly a opravy zautomatizovat. Standardy nebyly dodržovány nejčastěji z důvodu programování na různých verzích jednotlivých projektů, které používají různé standardy. Nejdříve bylo třeba prozkoumat veškeré možnosti, které mohou být pro kontrolu standardů použity. Použití veškerých nástrojů jsem konzultoval s hlavním architektem interních aplikací.

Výstupem byl seznam mnoha nástrojů pro různá odvětví kontroly zdrojového kódu. Ze začátku pro analýzu kódu nejlépe vycházel PHPStan, neboli PHP Static Analysis Tool. Jelikož však naše interní aplikace není postavena na žádném veřejně dostupném frameworku, bylo by nastavení tohoto nástroje natolik časově náročné, že byly vybrány jiné alternativy. Implementace tohoto nástroje se do budoucna však nevylučuje. Další nástroj, který nebyl použit se nazývá PHPCS Fixer. Ten dokáže špatné standardy pomocí správné konfigurace sám opravit a ten nebyl použit také z důvodu velké časové náročnosti konfigurace. Stěžejními použitými nástroji

jsou PHP_CodeSniffer, Parallel Lint, PHP Mess Detector a PHP Copy/Paste Detector. Veškeré kontroly jsou spouštěny pomocí nástroje Phing.

Výsledkem implementace těchto nástrojů byla získána přímá kontrola nad stylem, jakým je zdrojový kód psaný. Nemělo by se tedy stát, aby každý používal rozdílný styl psaní kódu.

5.4.1 PHP_CodeSniffer

Pro kontrolu standardů zdrojového kódu byl použit nástroj PHP_CodeSniffer. Ten ve výchozím stavu kontroluje zdrojové kódy jazyka PHP pomocí standardů PSR. Jejich součástí je například používání mezer místo tabulátorů pro odsazení nebo psaní složených závorek metod na samostatný řádek [11]. Tento styl byl pro interní aplikaci nevyhovující, a proto bylo nutné vytvořit vlastní konfiguraci pomocí souboru ve formátu XML. Vytvoření vlastní konfigurace je komplikované, jelikož nikde neexistuje žádná dokumentace k jednotlivým pravidlům používaných v konfiguraci. Musela být tedy procházena různá fóra či návody, a nakonec bylo vybráno zhruba 50 důležitých pravidel pro kontrolu. Mezi ty se řadí třeba správné odsazení, mezery za čárkou, mezery před složenými závorkami a další. Tato pravidla byla vybrána na základě požadavků hlavního architekta interních aplikací.

Výsledek kontroly může být programátorem přečten přímo v příkazovém řádku po spuštění přes Phing nebo v textovém souboru ve složce */build/reports/* nebo po automatickém spuštění pomocí Gitlab CI. Výsledek je možno vidět na výpisu 3. PHP_CodeSniffer může být spouštěn jednak pro celou aplikaci nebo pro aktuálně změněné soubory nebo pro všechny změněné soubory v rámci jedné úpravy. Velkou výhodou PHP_CodeSnifferu je, že je podporován přímo vývojovým prostředím PhpStorm. Programátor je poté na chyby ve standardech upozorňován přímo během psaní zdrojového kódu.

```
FILE: Category/CategoryDbMapper.php
----------------------------------------------------------------------
FOUND 2 ERRORS AFFECTING 2 LINES
----------------------------------------------------------------------
 39  | ERROR | [x] Expected 1 space(s) after IF keyword; 0 found

 131 | ERROR | [x] No space found after comma in function call

----------------------------------------------------------------------
PHPCBF CAN FIX THE 2 MARKED SNIFF VIOLATIONS AUTOMATICALLY
----------------------------------------------------------------------
```

Výpis 3: Výstup kontroly pomocí PHP_CodeSniffer

5.4.2 Parallel Lint

Nevýhodou jazyka PHP je, že není kompilovaný, ale pouze skriptovací. Programátor bez důkladného testování všech scénářů nemusí na chybu syntaxe vůbec přijít. Pro tyto účely byl vybrán nástroj Parallel Lint. Ten po spuštění kontroluje, zda kód neobsahuje syntaktické chyby. Výsledky kontroly syntaxe je možné vidět v textovém souboru nebo v příkazovém řádku ve tvaru viz výpis 4. U tohoto nástroje se nenastavují žádná pravidla pro kontrolování. Veškerá konfigurace je řešena v spouštěném příkazu. Alternativou k tomuto nástroji je automatické akceptační testování. Hlavní nevýhoda automatického akceptačního testování je v tom, že je tyto testy nutné pro každý scénář manuálně vytvořit.

```
Checked 238 files in 0.8 seconds
--------------------------------------------------
Parse error: CategoryRepository.php:14
    13| $mainCategory = array()
   >14| $categoryId = 0;
Parse error
```

Výpis 4: Výstup kontroly syntaxe pomocí Parallel Lint

5.4.3 PHP Mess Detector

Další nástroj, který byl vybrán byl PHP Mess Detector. Ten pomocí jednoho příkazu dokáže analyzovat zdrojový kód jazyka PHP. Ve zdrojovém kódu hledá možné logické chyby, příliš komplikované bloky kódu, nepoužívané metody či délky jednotlivých metod a tříd. V ukázkovém výpisu 5 je možné vidět upozornění na dlouhou třídu, která přesahuje předem definovaný počet řádků. Stejně jako v případě PHP_CodeSnifferu je třeba jej nakonfigurovat pomocí jednoho XML souboru. V konfiguračním souboru jsou veškeré pravidla, které má Mess Detector kontrolovat. Obrovská výhoda oproti PHP_CodeSnifferu byla v tom, že veškeré pravidla jsou řádně zdokumentována [12]. Není třeba tedy složitě pravidla hledat po různých internetových fórech.

```
CategoryDomainRepository.php 3 ExcessiveClassLength: The class
    CategoryDomainRepository has 203 lines of code.
Current threshold is 201. Avoid really long classes.
```

Výpis 5: Výstup kontroly pomocí PHP Mess Detector

5.4.4 PHP Copy/Paste Detector

Tento nástroj dokáže v předem definovaných složkách hledat, zda se v zdrojovém kódu nenacházejí duplikace kódu. U tohoto nástroje není taktéž třeba žádné složité konfigurace. Veškerá konfigurace je řešena v rámci spouštěného příkazu. Ve výchozím stavu je kontrolováno, zda neexistuje 5 stejných řádků kódu. Výsledek kontroly duplicitního kódu je vypsán do textového

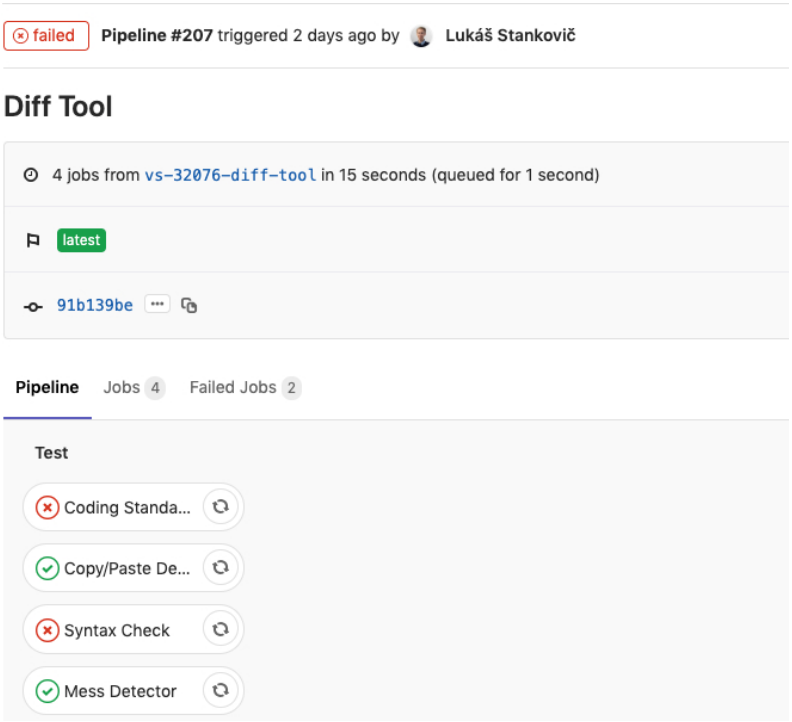
souboru, nebo do příkazového řádku. Ukázkový případ nalezení duplikací je možné vidět na výpisu 6. Je zde znázorněno, že v souborech *main.php* jsou řádky 70-114 a 52-96 duplicitní. Tento test je taktéž spouštěn při každém *commitu* pomocí Gitlab CI.

```
Found 1 clones with 44 duplicated lines in 2 files:
  - /builds/aplikace-vizitek/domain_admin/pages/index/main.php:70-114
    /builds/aplikace-vizitek/domain_card/pages/page/index/main.php:52-96
0.17% duplicated lines out of 25629 total lines of code.
```

Výpis 6: Výstup kontroly pomocí PHP Copy/Paste Detector

5.4.5 Gitlab CI

Pro automatické spouštění kontrol bylo rozhodováno mezi nástroji Jenkins a Gitlab CI. Aby nebyla použita další aplikace třetí strany byl vybrán Gitlab CI a ten pomocí Dockeru a Phingu spouští předem definované příkazy. Tyto příkazy jsou nakonfigurovány ve speciálním *yaml* souboru *.gitlab-ci.yml*. Veškeré kontroly se spouští automaticky po každém *commitu*, které programátor provede. Jednotlivé proběhlé kontroly jsou poté graficky zobrazeny pomocí tzv. *pipeline* viz obrázek 3. Pokud kontrola proběhla v pořádku, je uživateli zobrazena vedle *commitu* zelená ikonka. Jestliže kód obsahuje chyby, je zobrazena červená ikonka a zároveň je programátorovi poslán varovný e-mail, ve kterém jsou vypsány veškeré vyskytující chyby. Po kliknutí na detail kontroly je viditelný i příkazový řádek se spouštěnými příkazy na serveru.



Obrázek 3: Gitlab CI - neúspěšná pipeline

5.4.6 Phing

Základním implementovaným nástrojem pro spouštění všech kontrol zdrojového kódu je Phing [10] a ten dokáže automaticky spouštět příkazy z příkazového řádku. Výhodou je, že programátor nemusí znát přesné znění příkazů, které zároveň mohou mít i několik složitých parametrů. Další výhodou je, že pomocí jednoho příkazu lze spouštět vícero příkazů najednou. Konfigurace je přehledně uložená v kořenové složce projektu v souboru ve formátu XML. V konfiguraci je možné nastavit deklarace jednotlivých příkazů a proměnných používaných v úlohách.

Lze spouštět buď pomocí příkazu *phing*, nebo jedním kliknutím na tlačítko ve vývojovém prostředí PhpStorm. Phing se dále spouští při nasazování nové úpravy na produkční server nebo pouze při jakémkoliv nahrání nové změny do verzovacího systému Git. V tomto případě jsou příkazy Phing spouštěny automaticky pomocí Gitlab CI. Veškeré kontroly kódu by měly být před každým *commitem* programátorem na svém lokálním prostředí manuálně spouštěny.

6 Agregátor

6.1 Diff Tool

Jelikož se velmi často stává, že například Heureka změní specifikaci svých feedů, bylo mým dalším úkolem vytvořit modul *Diff Tool* do interní aplikace *Agregátor*, který bude tyto specifikace a dokumentace různých dodavatelů v pravidelných intervalech kontrolovat. Úkolem bylo, aby definované prvky na daných stránkách byly modulem kontrolovány jednou denně. Pokud prvek v předchozí kontrole neexistoval, nebo došlo k přidání nového vnitřního prvku, je modulem posláno upozornění na přednastavený e-mail.

Nejprve bylo nutné vymyslet, jakým způsobem budou změny na stránkách kontrolovány. Byl tedy proveden výzkum, jaký způsob bude použit. Bylo rozhodnuto pro stahování obsahu stránky použít funkce *curl()*. Důvodem pro zvolení této funkce bylo, že na rozdíl od jiných funkcí podporuje problém s přesměrováním. Další výhoda také je, že funkce pracuje bez problému i s protokolem HTTPS. Pro tuto funkcionalitu byla také vytvořena třída, která funkci zaobaluje a tím je možno ji jednoduše vícenásobně použít bez jakéhokoli nastavení. Další alternativou je využití například funkce *file_get_contents()*.

Byla vytvořena administrace, ve které je možné nastavit URL kontrolované stránky, selektor objektu, který se bude na stránce hlídat a seznam e-mailů, na které se bude posílat informace o případné změně. V administraci je také zobrazováno, kdy a kde došlo k nějaké změně stránky. Následně byla navržena databáze, se třemi tabulkami. Hlavní tabulka obsahuje základní informace o kontrolované stránce, jako například název, datum přidání či URL kontrolované stránky. Další tabulka obsahuje seznam e-mailů pro jednotlivé stránky a na tyto e-maily jsou posílány informace o změněné struktuře dané stránky. Poslední tabulka obsahuje log pro jednotlivé změny na webové stránce. Každá tabulka má svou entitu, servisní vrstvu, repozitář a databázový mapper.

Po dokončení administrace, byla započata implementace automatického modulu. Modul si nejprve z databáze pomocí databázového mapperu vybere, na jaké webové stránky má být napojen a jaké prvky na stránce mají být hledány. Potom jsou v dávkách procházeny již samotné stránky, kde jsou hledány definované prvky. Obsahy těchto prvků jsou kontrolovány s obsahem stáhnutým během poslední změněné kontroly. Pokud je obsah změněn, je ihned přidána do databáze nová verze obsahu. Zároveň jsou modulem do paměti uloženy informace o změněných prvcích. Tyto informace jsou poté hromadně poslány e-mailem. Změněné prvky na stránce jsou taktéž uloženy do souboru pro lepší pozdější porovnávání. Modul je spouštěn automaticky každý den. Pokud by došlo k záseku na dané stránce, například vinou nefunkční stránky, mohl by se zaseknout i náš modul. Z toho důvodu bylo použito pro kontrolu dávkování. Pokud dojde k danému záseku, jsou další stránky zkontrolovány v rámci další dávky. Administrátorovi je zároveň poslán e-mail o nefunkční skenované stránce.

Pro samotnou kontrolu daných selektorů na stránce byla využita veřejně dostupná knihovna

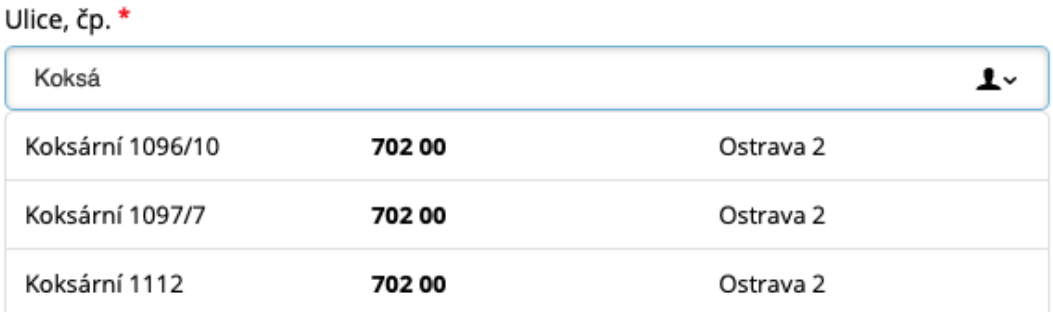
PHP Html Parser [13]. Pomocí této knihovny lze vytáhnout jakýkoli DOM objekt ze zadané stránky. DOM objekt je možné stáhnout buď jako text, nebo jako DOM objekt, s kterým lze dále pohodlně objektově pracovat a případně měnit staženou DOM strukturu. U knihovny lze nastavit i striktní kontrolu validního HTML dokumentu. Pokud by HTML dokument nebyl validní je vyhozena výjimka *StrickException*.

6.2 Napojení na katastr nemovitostí

Aktuálně je v současných e-shopech implementováno vlastní staré již plně nevyhovující řešení, které automaticky napovídá uživateli adresu ve formulářových polích k tomu určených. Toto napojení není již z důvodu chybějící pravidelné aktualizace adres vyhovující. Byla tedy vytvořena nová interní aplikace, sloužící pro ukládání a pravidelnou obnovu českých adres. Aplikace je postavena na novém jádru *Všeaplikace*.

Jednotlivé české adresy jsou aplikací stahovány z českého katastru nemovitostí ve formátu CSV [14]. Tyto adresy jsou stahovány každý měsíc. Jakmile je soubor do úložiště aplikace stažen, je ihned namapován na databázovou tabulku. Také je tento soubor převeden z kódování *CP1250* na *UTF-8* a zálohován do úložiště aplikace. Z důvodu úspory výpočetního výkonu je soubor, který je jednotlivými e-shopy stahován pro aktualizaci jejich databáze adres předgenerován. Soubor ve formátu JSON je uložen na speciální adrese, která je dostupná pouze na základě ověření hlaviček HTTP požadavku.

Po vygenerování JSON souboru v interní aplikaci je tento soubor stahován jednotlivými e-shopy. Soubor je automaticky do aplikací získáván pomocí servisního skriptu. Celý soubor je do e-shopu ukládán jednak z rychlostních důvodů při načítání položek pro napovídání uživateli, ale také z důvodu zálohy adres v případě, že interní aplikace pro generování souboru s adresami selže. Jestliže uživatel zadá alespoň část adresy do formulářového pole, je mu adresa postupně napovídána. Příklad doplňování adresy je možné vidět na obrázku 4. Pokud uživatel klikne na jakoukoli adresu ve výpisu doporučených adres, je adresa kompletně přepsána do formulářových polí. Touto funkčností je jednak uživateli urychlen a zjednodušen například objednávkový proces. Také se funkčností předejde překlepům v adresách.



Obrázek 4: Napovídání adres v objednávkovém procesu

6.3 Export a import dat

Další požadavek byl na vytvoření nového modulu pro e-shopy, který dovolí hromadně aktualizovat určitá data nastavení e-shopu. Pokud dojde momentálně k hromadné změně určité hodnoty nastavení, je třeba tuto hodnotu manuálně změnit na každém e-shopu zvlášť. Mezi tato data patří například změny URL adres pro různé SMS brány nebo například statická data pro kategorizaci Heureky a Google. Data je možné také posílat z e-shopu na agregátor. Pro správný přenos je nutné e-shop upravit a vytvořit na e-shopu odpovídající rozhraní pro API. Zabezpečení je vyřešeno na základě kontroly hlaviček a speciálních tokenů.

Veškerá data jsou přenášena pomocí REST API. To znamená, že data jsou posílána pomocí čtyř nejpoužívanějších HTTP metod. Pro získání zdroje z REST API je používána metoda *GET*, pro smazání zdroje je používána HTTP metoda *DELETE*. Pro vytvoření nových hodnot je používána metoda *POST* a pro aktualizaci již existujícího zdroje je běžně používána metoda *PUT* [15]. Celé API v aplikaci Agregátor je vytvořeno pomocí jazyka PHP. REST byl vybrán z důvodu jednoduchosti přenosu, kdy se pro každou akci volají jednotlivé URL se svou vlastní třídou. V těchto třídách jsou odbavovány veškeré požadavky.

Další možný způsob pro realizaci API bylo možné použít SOAP, který je založen na XML. Ten je na rozdíl od REST standardizovaný. REST zpravidla pro komunikaci používá formát JSON.

7 Závěr

Díky této praxi jsem nabyl mnoha praktických zkušeností a rozšířil si obzory v neustále měnícím se oboru informačních technologií. Velkým přínosem pro mne byla také veškerá kontrola zdrojového kódu zkušeným programátorem, který mi vždy vysvětlil, jakých chyb jsem se dopustil a čeho bych se měl vyvarovat. Během praxe jsem uplatnil teoretické znalosti získané během studia. Konkrétně se jedná zejména o znalosti z předmětů *Vývoj internetových aplikací*, *Úvod do softwarového inženýrství* a *Vývoj informačních systémů*. Také jsem získal nové dovednosti zejména v jazyce PHP a v oblasti vývoje internetových aplikací. Věřím, že veškeré tyto nabyté praktické znalosti uplatním v budoucnu při ucházení se o zaměstnání či ve vlastním podnikání.

Výsledkem mé praxe je především nová interní aplikace, která bude sloužit oddělením napříč celé firmy. Součástí této interní aplikace je menší aplikace Agregátor, která bude pomáhat především programátorům se změnami v různých napojení a tím jim ušetří drahocenný čas. Během praxe jsem si také vyzkoušel práci v týmu několika lidí s použitím verzovacího nástroje Git a měl možnost ovlivňovat samotný vývoj. Díky přínosným školením jsem získal dovednosti nejen z oblasti vývoje internetových aplikací, ale i z oblasti spojené s komunikací se zákazníkem.

Literatura

[1] Co je Shopsys Framework. *Shopsys* [online]. Ostrava: Shopsys, c2003-2019 [cit. 2019-02-17]. Dostupné z: https://www.shopsys.cz/co-je-shopsys-framework

[2] What is PHP?. *PHP* [online]. PHP Group, c2001-2019 [cit. 2019-03-10]. Dostupné z: http://php.net/manual/en/intro-whatis.php

[3] What is MySQL?. *MySQL* [online]. Oracle Corporation, c2019 [cit. 2019-03-11]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[4] What is HTML?. *W3Schools* [online]. Refsnes Data, c1999-2019 [cit. 2019-03-10]. Dostupné z: https://www.w3schools.com/whatis/whatis_html.asp

[5] Getting started | Less.js. *Less.js* [online]. Alexis Sellier, c2009-2019 [cit. 2019-03-14]. Dostupné z: http://lesscss.org/

[6] What is JavaScript?. *W3Schools* [online]. Refsnes Data, c1999-2019 [cit. 2019-03-10]. Dostupné z: https://www.w3schools.com/whatis/whatis_js.asp

[7] JQuery. *JQuery* [online]. jQuery Foundation, c2019 [cit. 2019-03-13]. Dostupné z: https://jquery.com

[8] About Git. *Git* [online]. Git [cit. 2019-02-27]. Dostupné z: https://git-scm.com/about/

[9] XML Path Language (XPath). *W3C* [online]. W3C, 1999 [cit. 2019-03-16]. Dostupné z: https://www.w3.org/TR/1999/REC-xpath-19991116/

[10] Phing. *Phing - a PHP build tool* [online]. c2018 [cit. 2019-03-16]. Dostupné z: https://www.phing.info

[11] PSR-2: Coding Style Guide. *PHP-FIG* [online]. PHP Framework Interop Group, c2019 [cit. 2019-03-16]. Dostupné z: https://www.php-fig.org/psr/psr-2/

[12] Current Rulesets. *PHP Mess Detector* [online]. Manuel Pichler, 2017 [cit. 2019-03-12]. Dostupné z: https://phpmd.org/rules/index.html

[13] PHP Html Parser. *Github* [online]. c2019 [cit. 2019-03-24]. Dostupné z: https://github.com/paquettg/php-html-parser

[14] Adresní místa RÚIAN ve formátu CSV. *Nahlížení do katastru nemovitostí* [online]. Praha: Český úřad zeměměřický a katastrální, c2004-2019 [cit. 2019-03-15]. Dostupné z: https://nahlizenidokn.cuzk.cz/StahniAdresniMistaRUIAN.aspx

[15] RICHARDSON, Leonard a Michael AMUNDSEN. *RESTful Web APIs*. Beijing: O'Reilly, 2013, s. 33. ISBN 978-1-449-35806-8.